VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTERAKTIVNÍ STAVEBNICE POMOCÍ OPENGL

INTERACTIVE BUILDING GAME WITH OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **KAROLÍNA KLEPÁČKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL TÓTH**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Klepáčková Karolína**
Obor: Informační technologie
Téma: **Interaktivní stavebnice pomocí OpenGL**
Interactive Building Game with OpenGL
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte knihovnu OpenGL a její nadstavby.
2. Navrhněte aplikaci umožňující interaktivní práci se stavebnicí typu LEGO.
3. Implementujte navrženou aplikaci
4. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu
5. Vytvořte video pro prezentování projektu.

Literatura:

- Podle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3
- Funkční prototyp aplikace

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Tóth Michal, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této bakalářské práce je implementace interaktivní stavebnice typu LEGO® pomocí OpenGL. Aplikace umožňuje tvorbu modelů pomocí kostek, u nichž lze měnit barva a typ. Vytvořené modely lze ukládat i načítat. Každá kostka navrženého modelu má svou kombinaci typu a barvy. Výstupem je údaj o počtu kostek každé kombinace, která se v aplikaci nachází. Díky tomu lze navržený model následně realizovat. Výsledná implementace je uskutečněna pomocí jazyka C++ rozšířeného o knihovny.

Abstract

The goal of this thesis is implementation of interactive LEGO® like building game using OpenGL. Aplication allows to create models using bricks which color and type can be changed. Created models can be saved or loaded. Each brick of a designed model has its own combnation of type and color. The output is number of bricks of each combination which occurs in application. This enables to realize the designed model. The final implementation is realized via C++ programming language extended with libraries.

Klíčová slova

Interaktivní stavebnice, OpenGL, C++, 3D, LEGO, výběr myší, detekce kolizí, zobrazování modelů, ImGUI, stínování, osvětlovací modely, graf scény.

Keywords

Interactive building game, OpenGL, C++, 3D, LEGO, mouse picking, collision detection, models rendering, ImGUI, shading, lighting models, scene graph.

Citace

KLEPÁČKOVÁ, Karolína. *Interaktivní stavebnice pomocí OpenGL*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Tóth Michal.

Interaktivní stavebnice pomocí OpenGL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Michala Tótha. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Karolína Klepáčková
16. května 2017

Poděkování

Ráda bych zde poděkovala vedoucímu práce Ing. Michalu Tóthovi za odborné vedení a celému týmu volitelného semináře z grafiky za cenné rady.

Obsah

Kapitola 1

Úvod

Každý z nás byl jednou malý a chtěl dělat něco kreativního. Ne všichni však měli svoji vlastní hračku, pomocí které by mohli rozvíjet tvořivé myšlení. Avšak díky moderním technologiím jsou tyto možnosti dostupné pro drtivou většinu z nás.

V dnešní době je zvykem, že děti již od útlého věku více interagují s počítačovou technologií než s reálnými objekty. Ať už skrze stolní počítače, notebooky nebo mobilní zařízení. Jejich rozvoj je proto odlišný od generací, které vyrůstaly dříve a přišly do styku s větším počtem fyzických hraček než s elektronickými, nehmatatelnými hrami.

Dnes existuje nespočetné množství aplikací pro rozvoj myšlení, kreativity a prostorové představivosti. Může se jednat o hádanky, hlavolamy, interaktivní stavebnice a další jim podobné. Mnohé z her tohoto typu si našlo cestu k našim srdcím, avšak stavebnice typu LEGO® jsou nadčasové a oblíbené napříč všemi generacemi a nezáleží na tom, jak je tato hra zprostředkována. V současnosti nejsou tyto stavebnice pouze dětskou zábavou, ale existuje i mnoho dospělých lidí, které tento fenomén nikdy nepřestane bavit.

S trendem tvorby stavebnic pro širokou věkovou škálu zákazníků se ztotožnily i firmy, které tyto stavebnice vyrábějí. Díky tomu vznikají speciální sběratelské kusy, avšak největší nadšenci se ani s touto nabídkou nespokojí. Často chtějí více, než hotové stavebnice s pevně danými návody a kostkami. Proto hledají způsob, jak realizovat své vlastní představy.

V současnosti existuje více aplikací, ve kterých lze svůj model detailně navrhnout a spočítat potřebné kostky v konkrétních barvách. Takto využitelnou aplikaci poskytuje například i firma LEGO®, která nabízí možnost objednání přesného počtu kostek daných tvarů a barev podle požadavku zákazníka.

Cílem této práce je vznik prostředí pro kreativní vyžití a možnost realizace vlastních nápadů; aplikace umožňující tvorbu vlastního modelu pomocí kostek. Ty si uživatel vybírá z dostupného seznamu v grafickém uživatelském rozhraní, kde lze libovolně měnit jejich barva.

Výsledkem je interaktivní stavebnice, která umožňuje uživateli pohyb po herní ploše, přidávání, přemisťování a odebírání kostek.

Kapitola 2

3D svět v počítači

Vývoj 3D světa je poměrně náročný a komplexní proces. Vyžaduje nejen dobrý návrh a zpracování, ale hlavně kvalitní teoretický základ. Předpokládá se, že tuto práci budou číst čtenáři znalí základů programování a práce s grafickou knihovnou OpenGL, proto zde nebude podrobně probírána.

Hlavními body, kterými se zabývá tato kapitola, jsou teoretické poznatky potřebné pro implementaci 3D aplikace vytvořené pomocí OpenGL. Stěžejní znalosti, získané po přečtení, pomohou čtenáři s orientací v základních pojmech týkajících se prostředí 3D aplikace.

2.1 Trojrozměrný počítačový model

Neodmyslitelnými prvky interaktivních aplikací jsou objekty, se kterými uživatel interaguje. Zpravidla se setkáváme s aplikacemi, jejichž prvky mají předlohu v objektech reálného světa. Některé však poskytují herní prostředí, ve kterém se propojují reálné objekty se smyšlenými. Jako demonstrace je zde uvedena hra šachy oproti akční hororové hře Resident Evil 4.

Obrázek 2.1: Příklad hry s reálnými objekty: Šachy (www.geeky-gadgets.com).

Obrázek 2.2: Příklad hry se smyšlenými objekty: Resident Evil 4 (mafiagamezone.cz).

Zobrazení herní scény v počítači je vždy určitou optimalizací reality. Existují tři způsoby, jak jednotlivé objekty do aplikace zpropagovat. Způsoby jsou převzaty z knihy *Moderní počítačová grafika* [11, kapitola 8].

Modelování založené na obrazech

V anglické literatuře a na internetu se setkáme nejčastěji s označením Image based modeling. Tento způsob vychází z geometrie reálného objektu, kterou do počítače nahrajeme pomocí 3D skeneru nebo se vytváří z několika fotografií daného předmětu. Výsledkem je přibližný model reality, který nemusí mít dostatečnou kvalitu pro využití v aplikaci. K defektům a nepřesnostem modelu dochází v závislosti na velikosti, propracovanosti či členitosti objektu.

Modely tvořené člověkem

Tento postup je velmi pracný a náročný na zdroje. Aby vznikl kvalitní model, je zapotřebí mít talentovaného člověka se zkušenostmi s tvorbou 3D grafiky. Výsledný model bývá zpravidla přesný, s dostatečnou mírou detailů, neboť autor ví, co chce modelovat, a jak má která část modelu vypadat.

Procedurální generování modelů

Poslední možností jak vytvořit 3D model je procedurální modelování. Dá se rozdělit na dvě hlavní skupiny.

První jsou metody, které se využívají v CAD aplikacích (computer-aided design, volně přeloženo jako počítačem podporovaný návrh), které uživateli poskytují prostředí pro projektování. Jsou to metody jako šablonování, generování ploch z křivek a další.

Druhá skupina, kterou považujeme za „pravé“ procedurální modelování, se zaměřuje na generování objektů algoritmicky. Generované modely odpovídají vzhledem či chováním objektům, které běžně nacházíme v přírodě. Tato skupina se dále dělí podle několika různých kritérií. Existují např. algoritmy vycházející z gramatik – L-systémy (vhodné pro generování rostlin), algoritmy fraktální geometrie (pro generování krajin, kamenů, stromů atp.) a systémy částic (pro generování explozí, simulaci ohně). Jejich studiem se ovšem zabývá již mnoho publikací a není předmětem této závěrečné práce.

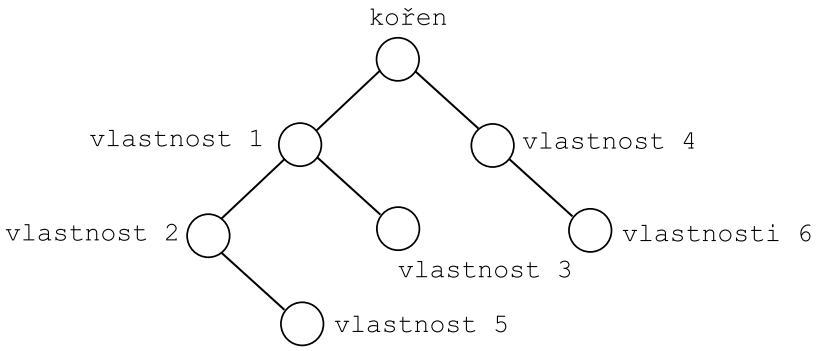
2.2 Scéna

Pojem scéna zahrnuje rozvržení objektů jedné aplikace. Komplexnější aplikace mohou mít hned několik scén. Na monitoru počítače můžeme v jednom okamžiku vidět celou scénu nebo jen její část. Hlavními prvky jsou objekty. Ty mají určitý postup zobrazení, který je nutno dodržet. Ke scéně samotné se pojí několik principů a technik, kterými dosáhneme jejího zdokonalení.

Graf scény

Obecně graf scény znázorňuje objekty v prostorové scéně. Má schopnost vyjádřit jejich vlastnosti a vztahy mezi nimi (viz obrázek 2.3). Jednou z nejdůležitějších vlastností je dědičnost. Ta nám umožňuje v konkrétním uzlu definovat vlastnost, která bude společná pro všechny následníky.

Grafem scény rozumíme jeden či více n-árních stromů. N-ární strom je graf, v nemž platí, že pro každý uzel existuje právě jeden předchůdce. Toto pravidlo neplatí pro kořen stromu, který v něm stojí na nejvyšší úrovni. Úrovně se v stromových strukturách počítají zpravidla od kořene, který je nejvýš a pokračují jednotlivými větvemi vertikálně k nižším vrstvám. Konkrétní typy uzlů stromu, pravidla pro jeho tvorbu i interpretace jeho dat závisí na systému, ve kterém je definován.



Obrázek 2.3: Graf scény, hiearchické zobrazení vzájemného vztahu vlastností objektů ve scéně. Vlastnosti definované v uzlech nadřazených jsou společné pro všechny následníky.

Jak již bylo zmíněno, vlastnosti se vždy zpropagují na uzly následující. Existují systémy, které využívají grafu scény jako určení pořadí následníků. Tato hiearchie umožňuje rozšířit působnost dědičnosti i v horizontálním směru, kupříkladu zleva doprava. Pak by to znamenalo, že všechny prvky napravo od nejlevějšího uzlu budou sdílet jeho vlastnost.

V závislosti na definici grafu scény se aplikace jednotlivých vlastností na objekt liší. Výsledná vlastnost se buď skládá ze všech předcházejících vlastností, jak je tomu například u transformací. Tzn. výsledná vlastnost je součin všech předcházejících transformací. Kdybychom naopak měli jako vlastnost barvu, tak se na objekt aplikuje pouze poslední barva, která je v průchodu stromu od kořene k objektu nejblíže danému objektu.

Výhodou grafu scény je jednoduchost změny vlastností všech objektů jednoho podstromu zároveň, aniž by se musela vlastnost měnit každému objektu samostatně. Aby byl graf scény pro aplikaci přínosný, je nutné, aby bylo předem promyšleno, jaké objekty se v ní budou nacházet, a jak spolu budou hiearchicky souviset. Podrobnějším popisem grafu scény se zabývá jedna z kapitol knihy *Moderní počítačová grafika* [11, podkapitola 14.1].

2.3 Osvětlovací modely

Osvětlovacími modely (anglicky Lighting models) rozumíme matematické modely, pomocí nichž se vypočítávají barvy objektu ve scéně v závislosti na dopadajícím světle. Jsou vždy definovány odrazovou funkcí. Ta nám udává výslednou intenzitu odraženého světla v jednom bodu objektu. Její hodnota záleží na intenzitě a poloze zdroje světla, vlnové délce paprsku, směru odrazu paprsku a barvě objektu. Světlo jako takové nám přináší hloubkovou informaci o poloze a tvaru objektů. Podle míry složitosti dělíme osvětlovací modely na dvě základní skupiny, jimiž jsou empirické a realistické modely.

- Empirické

  Modely, které počítají s jednodušším zobrazením reality. Jsou výpočetně rychlé a používají se v real-time renderování.

- Realistické

  Výpočetně složité modely s vysokou mírou přesnosti. Tyto modely věrně reflektují realitu. Používají se při vytváření vysoce realistických scén, kde nezáleží na rychlosti zpracování, nýbrž na kvalitě výsledku.

Obecně platí, že povrch objektu je složen z velkého množství odrazových plošek (polygonů). Každá z plošek odrazí paprsek s určitou intenzitou, který při odrazu modifikuje. Pokud jsou plošky uspořádány jedním směrem, tak tvoří lesklé nebo přímo zrcadlové povrchy. Tohoto efektu lze dosáhnout korelací a natočením odrazových plošek. Naopak je tomu, pokud jsou plošky natočeny náhodně. Pak tvoří difúzní či-li matné povrchy, na nichž se světlo rozptyluje do mnoha směrů.

Konstantní barva

Nejjednodušší způsob, jak vykreslit objekt, je pomocí konstantní barvy. Tento způsob není přímo osvětlovacím modelem, avšak je vhodné uvést, proč je pro 3D grafiku nevhodný. Objekt, který je vykreslen tímto způsobem, ztrácí jakoukoliv hloubkovou informaci o svém vlastním tvaru, a jakékoliv materiálové vlastnosti jsou potlačeny. Použitím konstantní barvy se opticky vytvoří z 3D modelu pouze 2D model. Proto je pro 3D grafiku tento přístup nepoužitelný.

Lambertův osvětlovací model

Patří k nejjednodušším empirickým osvětlovacím modelům. Jeho výpočet je založen na jediné difúzní složce. Při výpočtu výsledné intenzity odraženého světla se zanedbává pozice pozorovatele (jeho směr pohledu). Díky tomu je výsledná intenzita ze všech úhlů pohledu na scénu stejná. Model je popsán normálovým vektorem, vektorem dopadu světla, intenzitou dopadajícího světla a koeficientem difuze. Koeficient difuze udává míru odrazivosti materiálu. Skalární součin normálového vektoru a vektoru dopadu světla je ekvivalentní s kosinem úhlu mezi těmito vektory. Ten udává, kolik světla se od daného polygonu objektu odrazí.

Nejlepší výsledky dává tento model, pokud jsou objekty, které pomocí něj zobrazujeme, vysoce teselované[1]. Výsledné objekty se skládají z velkého počtu trojúhelníků. Osvětlovací

[1]Teselace - pokrytí roviny geometrickými útvary (polygony). V případě 3D grafiky to bývají zpravidla trojúhelníky. Ty se nepřekrývají a vyplňují rovinu bez mezer. Detailnější popis a více zdrojů lze dohledat na internetové stránce [17].

model se počítá pro každý z těchto trojúhelníků a plynulost světelného přechodu roste s mírou teselace objektu.

Lambertův osvětlovací model je vhodný pro bezodleskové osvětlování, kde záleží na rychlosti výpočtu a nikoliv na míře realističnosti zobrazení.

Pro určení intenzity odraženého světla se používá Lambertův kosinový zákon viz [11, str. 335], pro lepší představu znázorněný na obrázku 2.4:

$$I_D = I_L \cdot r_D \cdot \cos\phi, \tag{2.1}$$

$$I_D = I_L \cdot r_D \cdot (\vec{N} \cdot \vec{L}). \tag{2.2}$$

Jednotlivé proměnné ve vzorci 2.1 určují:

I_D - Intenzita odraženého světla,

I_L - Intenzita dopadajícího světla,

r_D - Koeficient difuze (odrazivosti materiálu),

ϕ - Úhel dopadu paprsku na povrch objektu.

Zníměný vzorec 2.1 lze převést na vzorec 2.2, kde:

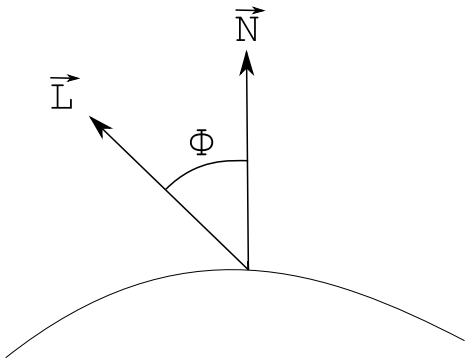
I_D - Intenzita odraženého světla,

I_L - Intenzita dopadajícího světla,

r_D - Koeficient difuze (odrazivosti materiálu),

$\vec{N}$ - Normálový vektor,

$\vec{L}$ - Vektor dopadu světla.



Obrázek 2.4: Grafické znázornění výpočtu Lambertova osvětlovacího modelu. ϕ určuje úhel dopadu paprsku na povrch objektu. Normálový vektor je označen $\vec{N}$. Vektor dopadu světla je označen $\vec{L}$.

Výsledkem je intenzita světla (barva) jednoho trojúhelníku objektu. Vztah 2.2 je smysluplný pouze pro hodnoty

$$\vec{N} \cdot \vec{L} > 0,$$

tzn. pouze pro trojúhelníky, které jsou „nasvětleny“. V opačném případě je výslednek nulový, tzn. trojúhelníky jsou odvráceny od světla. Čím více je směr dopadu bližší k normálovému vektoru, tím větší je výsledné množství odraženého světla. Lambertův model je dále využíván i v jiných osvětlovacích modelech, kde zastává funkci difúzní složky.

Phongův osvětlovací model

Empirický osvětlovací model, jenž nese jméno autora, kterým je pan Bui Tuong Phong. Publikoval jej v roce 1977 ve svém článku v magazínu *Communications of the ACM* [10]. Detailnější popis lze nalézt v jeho odborné práci z roku 1973 [9].

V dnešní počítačové grafice je ze všech modelů využívaný nejvíce. Prvenství si získal hlavně rychlým výpočtem a kvalitní reprezentací osvětlení velkého množství polygonů (trojúhelníků).

Celkově je tento model komplexnejší, než Lambertův osvětlovací model. Výpočet je založen na součtu tří složek, jimiž jsou:

1. Ambientní složka

   Konstantní složka, která nahrazuje veškeré sekundární zdroje světla[2] ve scéně.

   Vypočítá se pomocí vztahu:

   $$I_A = I_a \cdot r_A = I_a \cdot r_D, \tag{2.3}$$

   kde I_A určuje výslednou ambientní složku, I_a udává množství okolního světla a r_A je barevný koeficient. Ten udává schopnost objektu odrážet okolní světlo. Je téměř totožný s r_D (viz rovnice 2.2), proto jsou zaměnitelné.

   Díky této složce, která má na celkové osvětlení minimální význam, je zajištěna odlišitelnost objektu od pozadí. Její zásluhou vidíme i strany objektu, které jsou odvráceny od světla, mírně světlejší než je pozadí.

2. Difúzní složka

   Difúzní složkou Phongova osvětlovacího modelu je Lambertův osvětlovací model. Ten se spočítá pomocí vztahu $I_D = I_L \cdot r_D \cdot (\vec{N} \cdot \vec{L})$ (detailněji viz rovnice 2.2). Všechny zákony platící pro Lambertův osvětlovací model platí i pro difúzní složku Phongova osvětlovacího modelu. Spolu se spekulární (zrcadlovou) složkou jsou vypočítávány z jednoho či více primárních zdrojů světla ve scéně.

3. Spekulární (zrcadlová) složka

   Spekulární složka zde znázorňuje odlesk materiálu objektu. Množství světla odraženého touto složkou je maximální, pokud směr odrazu světla je blízký směru pohledu.

   Dá se vyjádřit jako:

   $$I_S = I_L \cdot r_S \cdot \cos\alpha, \tag{2.4}$$

   přičemž:

   I_S - Intenzita odraženého světla,

   I_L - Intenzita dopadajícího světla,

   r_S - Koeficient zrcadlového odrazu,

   α - Úhel mezi vektory $\vec{V}$ a $\vec{R}$.

[2] Sekundární zdroj světla – zdroj světla, který vzniká odražením paprsku z primárního zdroje světla od objektu ve scéně.

Vzorec 2.4 je ekvivalentní se zápisem:

$$I_S = I_L \cdot r_S \cdot (\vec{V} \cdot \vec{R})^{n_s}, \tag{2.5}$$

kde:

I_S - Intenzita odraženého světla,

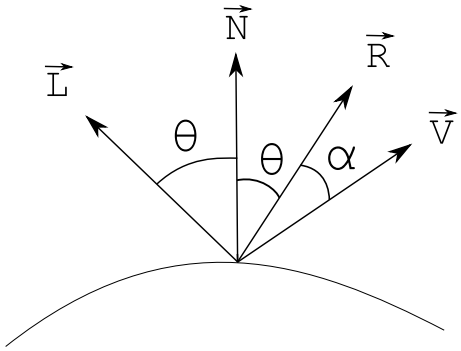
I_L - Intenzita dopadajícího světla,

r_S - Koeficient zrcadlového odrazu,

$\vec{V}$ - Vektor pohledu,

$\vec{R}$ - Vektor ideálního zrcadlového odrazu,

n_s - Koeficient ostrosti.



Obrázek 2.5: Znázornění složek Phongova osvětlovacího modelu. $\vec{L}$ je vektor dopadu světla. Normálový vektor je značen $\vec{N}$. $\vec{R}$ je vektor ideálního zrcadlového odrazu. Vektor pohledu $\vec{V}$ určuje směr k pozorovateli.

Vektor $\vec{R}$ je symetrický k vektoru $\vec{L}$ podle normálového vektoru $\vec{N}$. Proto jej lze vypočítat pomocí vzorce $\vec{R} = 2 \cdot (\vec{L} \cdot \vec{N}) \cdot \vec{N} - \vec{L}$. Pokud je skalární součin vektorů $\vec{V}$ a $\vec{R}$ menší než 0, tak víme, že je pozorovatel ve stejné části prostoru jako zdroj světla. Díky tomu nemůže pozorovatel vidět žádný odraz. Koeficient ostrosti n_s se udává v rozmezí $\langle 1, \infty \rangle$. Čím větší tento koeficient je, tím menší a ostřejší odlesky budou. Ideální zrcadlo by mělo $n_s = \infty$.

Kompletní vzorec pro Phongův osvětlovací model obsahuje všechny zmíněné složky. Jednoduše se dá zapsat jako:

$$I_P = I_A + I_D + I_S, \tag{2.6}$$

a po dosazení jednotlivých složek:

$$I_V = I_a \cdot r_A + \sum_{k=1}^{M} I_{L_k} \cdot [r_S \cdot (\vec{V} \cdot \vec{R_k})^{n_S} + r_D \cdot (\vec{L_k} \cdot \vec{N})]. \tag{2.7}$$

Vzorec 2.7 je obecný pro 1 až M světelných zdrojů ve scéně. Pro výslednou intenzitu osvětlení jednoho bodu povrchu počítáme ambientní složku pouze jednou, neboť svým významem zastupuje globální osvětlení scény. Složku difúzní a spekulární počítáme pro každý zdroj světla, který námi počítaný bod ovlivňuje.

Dvousměrová odrazová distribuční funkce

Tato funkce, zkráceně označována jako BRDF (anglicky Bidirectional Reflectance Distribution Function), udává odrazovou vlastnost materiálu objektu v konkrétním bodě. Je to realistický, fyzikálně založený model. Díky míře realističnosti je výpočetně náročný. Vzorec výpočtu:

$$f_r(x, \vec{w_r}, \vec{w_i}) = \frac{d\vec{L_r}(x, \vec{w_r})}{d\vec{L_i}(x, \vec{w_i})(\vec{w_i} \cdot \vec{n}), d\vec{w_i}}. \tag{2.8}$$

Ekvivalentním zápisem vzorce 2.8 je:

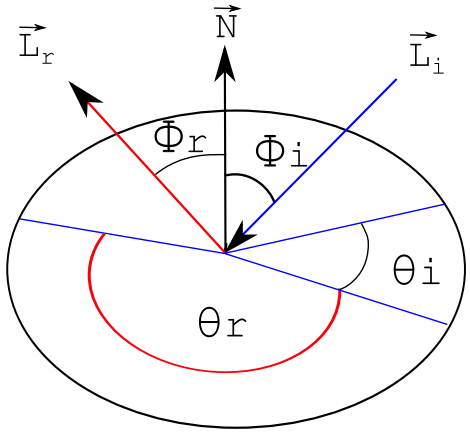
$$f_r(P, \vec{w_i}, \vec{w_r}) = \frac{\vec{L_i}(P, \vec{w_i})}{\vec{L_r}(P, \vec{w_r}) \cdot \cos\phi_i \cdot dw_i}, \tag{2.9}$$

kde:

$\vec{L_i}(P, \vec{w_i})$ - Ozáření,

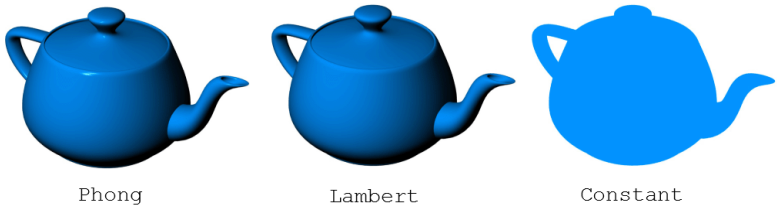
$\vec{L_r}(P, \vec{w_r})$ - Odražená zářivost,

$\vec{L_e}(P, \vec{w})$ - Vlastní zářivost ($\neq 0$, radiozita).

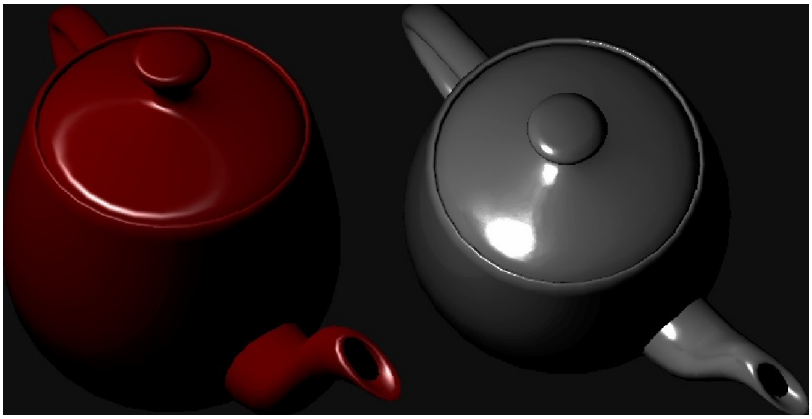


Obrázek 2.6: Grafické znázornění BRDF funkce. Vektor $\vec{L_r}$ určuje odraženou zářivost. $\vec{N}$ je normálový vektor. Vektor $\vec{L_i}$ vyjadřuje ozáření.

Pro podrobnější informace lze nahlédnout do knihy *Radiosity and Realistic Image Synthesis* [2]. Dá se říct, že BRDF je funkce, která udává, kolik energie se má transformovat ze vstupu na výstup podle úhlu, pod kterým dopadá vstupní paprsek. Výsledná intenzita je v rozsanu $\langle 0, 1\rangle$. Popisuje, kolik vstupní intenzity se odrazí, a jakým směrem, přičemž 0 vyjadřuje minimální (žádné) odražení a 1 maximální (úplné) odražení. Z této metody vychází všechny ostatní modely.



Obrázek 2.7: Srovnání Phongova osvětlovacího modelu – vlevo, Lambertova osvětlovacího modelu – uprostřed a konstantní barvy objektu – vpravo (developer.apple.com).



Obrázek 2.8: Objekty modelované pomocí BRDF. Imitují realitu věrohodněji, než osvětlovací modely použité na obrázku 2.7 (převzato z materiálu [7]).

2.4 Stínování

Stínování (anglicky shading) zahrnuje metody, které pomáhají určovat barvy všech pixelů modelu 3D scény. Udávají, pro které konkrétní body na povrchu modelu bude spočítán osvětlovací model, a které budou jen ovlivněny výpočtem, aniž by se výpočtu přímo účastnily. Existují metody, například Ray Tracing (blíže popsaný v knize *An Introduction to Ray Tracing* [3]), které počítají osvětlovací model pro všechny pixely ve scéně zvlášť. Tyto metody jsou zdlouhavé a výpočetně náročné. Optimalizací výpočtu pouze pro některé pixely dosáhneme rychlejšího vykreslování scény při současném zachování její poměrné realističnosti. Do jaké míry je vykreslení realistické, záleží na použité stínovací metodě.

Konstantní stínování

Konstantní stínování (anglicky flat shading) je jednou z nejjednodušších stínovacích metod. Nezohledňuje zakřivení povrchu modelu, avšak tato vlastnost nemusí být vždy problémem. Výsledná barva je konstantní a vykresluje se pomocí ní celý polygon, pro který je osvětlovací model počítán.

Tento model zvýrazňuje přechody mezi polygony a proto je vhodný například pro objekty v CAD aplikacích; technické modely součástek, konstrukcí a ostatní modely, kde je požadavek na zřetelně viditelné hrany. Povrch se jeví jako po částech lineární. Naopak pro obecná tělesa, kde se vyžaduje plynulost přechodů, je konstantní stínování nevhodné.

Metoda předpokládá, že každá plocha má jediný normálový vektor. Pokud není implicitně obsažena v datech prostorového modelu, lze ji u konvexních rovinných polygonů určit jako vektor, který bude ukazovat do vnějšího poloprostoru. Podle normálového vektoru je vypočítána barva středového pixelu, která je při rasterizaci přiřazena všem ostatním pixelům polygonu. Hlavním pozitivem této metody je snadná implementace v hardware a podporuje ji i OpenGL.

Goraudovo stínování

Stínovací metoda navržena panem Gouraudem v roce 1971 v článku *Continuous Shading of Curved Surfaces* [4]. V současné době se používá v grafických akcelerátorech, díky možnosti optimalizace na celočíselnou aritmetiku, která je dostatečně výpočetně rychlá. Nejvhodnější je použít Gouraudovo stínování pro modely, jejichž povrch je aproximován množinou polygonů. Výsledkem je plynulé stínování zakřivených obecných objektů, aproximace množinou polygonů není patrná.

Výpočet osvětlovacího modelu se v této metodě provádí pro každý vrchol polygonu. Proto je potřebné znát průměrný normálový vektor a barvu v každém z vrcholů zpracovávaného polygonu, aby bylo osvětlení modelu spočitatelné. Následně se barva vnitřních bodů vypočítá pomocí bilineární interpolace[3].

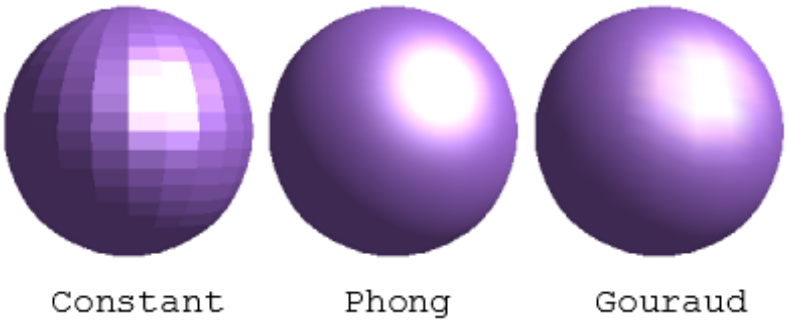
Osvětlovací modely jsou vyhodnocovány zvlášť pro ambientní a difúzní složku. Složka spekulární zde postrádá smysl díky interpolaci barev, která probíhá při rasterizaci. Barvu lze interpolovat po jednotlivých složkách, pokud ji máme ve tvaru trojsložkového vektoru (r, g, b). Složky z rozsahu $\langle 0, 1\rangle$ lze převést na číslo v celočíselném intervalu 0-255. V takovém případě se interpolace provede jako celočíselná operace, která je rychlejší a méně náročná, než výpočet v desetinných číslech.

Phongovo stínování

Jak již název napovídá, bylo navrženo stejným autorem jako Phongův osvětlovací model v článku *Illumination for Computer Generated Pictures* [10]. Ze zmíněných stínovacích metod vypadají její výsledky nejrealističtěji. Je určena k plynulému stínování obecných objektů, podobně jako Gouraudovo stínování. Dokáže vykreslit plynulý barevný přechod i při nízké teselaci objektu.

Narozdíl od Goraudova stínování je nutné znát normálové vektory nejen ve vrcholech, nýbrž po celé ploše polygonu. Pro výpočet normálových vektorů uvnitř polygonu se využívá bilineární interpolace normálových vektorů ve vrcholech. S těmito hodnotami je možné vypočítat osvětlovací model v každém pixelu stínovaného modelu. Díky tomu je Phongovo stínování výpočetně náročnější než konstantní a Gouraudovo stínování.

[3] Bilineární interpolace – medota spočívá v získání barvy pixelu pomocí čtyř okolních pixelů. Detailnější popis viz [11, podkapitola 22.6.4] nebo [13].



Obrázek 2.9: Srovnání konstantního stínování – vlevo, Phongova stínování – uprostřed a Gouraudova stínování – vpravo (https://henry2005.jimdo.com).

Kapitola 3

Interakce uživatele s aplikací

Aby bylo možné o aplikaci říct, že je interaktivní, je nutné tuto interakci uživateli zprostředkovat. Interakce s objekty aplikace se liší v závislosti na jejím typu. Potřebnými prvky pro tuto práci jsou výběr objektu (viz podkapitola 3.1), kolize (viz podkapitola 3.2) a ergonomie (viz podkapitola 4.9).

3.1 Výběr objektu

Jednou ze základních operací, kterou si lze představit u interaktivní aplikace, je výběr objektu. Základní možností, jak na počítači identifikovat objekt, je zpravidla možnost výběru pomocí klávesnice nebo myši.

V případě vybírání pomocí klávesnice lze aplikovat dva přístupy. Prvním je přiřazení jedné klávesy jednomu objektu. Druhým přístupem je použití seznamu, ve kterém lze objekt vybrat přepínáním pomocí vybraných kláves ve skupině objektů scény. Klávesnice je poměrně rychlým řešením pro scény, kde je zobrazeno malé množství objektů.

Avšak tato kapitola bude zaměřena především na výběr pomocí myši. Obecně je pro vybírání objektů vhodnějším prostředkem, neboť s ní lze pohodlně pracovat i ve scénách s velkým množstvím objektů. V praxi se pro výběr jednoho či více objektů ve scéně využívají různé přístupy, které se od sebe do značné míry liší. Konkrétně zde budou popsány tři z nich.

Výběr pomocí unikátní barvy

Tento výběr je založený na módu tzv. dvojitého bufferu. To znamená, že namísto vykreslení na obrazovku (zapsání pouze do jednoho renderovacího bufferu), je po zaměření objektu celá scéna překreslena do druhého bufferu, kde se každý objekt reprezentuje pomocí unikátní barvy zakódované bitovým posunem. Uživatel nikdy nevidí objekty vykreslené pomocí zakódované barvy. Tímto vznikne jedinečný identifikátor každého modelu, tzv. ID. Po zaměření se přečte barva nacházející se pod kurzorem a určí se pomocí ní konkrétní objekt. Na tomto společném principu jsou založeny dvě metody výběru.

- Pomocí zadního bufferu (back buffer)

  Největším úskalím tohoto způsobu výběru objektu je omezené množství identifikátorů. U velkých scén může dojít k vypotřebování všech kombinací unikátních barev. Také je důležité vědět, že při psaní přenositelného kódu je vhodné myslet na různorodost implementace identifikátorů. Příklady a podrobnější popis této metody lze najít v knize *OpenGL Průvodce programátora* [15, str. 505].

- Pomocí renderování mimo obrazovku (off-screen rendering)

  Modernějším přístupem je výběr pomocí renderování mimo obrazovku. Jeho výhodou oproti využití zadního bufferu je možnost vygenerování dostatečného počtu jedinečných ID. Založený je na použití off-screen framebufferu, jehož formátem je uint32[1]. Ten poskytuje 4 294 967 295 unikátních ID. Jejich množšství je dostačující pro drtivou většinu aplikací.

Výběr a zaměření pomocí OpenGL

OpenGL disponuje několika různými módy operací. Jedním z nich býval i mód výběru (GL_SELECT). V dnešních aplikacích se již nevyužívá (od OpenGL 3.0 je zakázaný). Poskytuje možnost určit konkrétní objekt na obrazovce, který byl uživatelem vybrán pomocí kurzoru.

Pokud máme aplikaci v módu výběru, tak se nám nemění obsah žádného z bufferů do té doby, než tento mód opět opustíme. V tomto se liší od renderování, které nám nově získané informace inhed zpropaguje do bufferů. Po opuštění výběrového módu vrátí OpenGL seznam objektů, které by uživatel mohl potencionálně na obrazovce vybrat. Tento seznam se nazývá záznam zásahů a odpovídá obsahu zásobníku jmen[2].

Rozšířením výběru je tzv. zaměření. Jedná se o proces, kterým lze zjistit jaká část scény, tj. který konkrétní objekt, je zaměřen například kurzorem myši. Použitím speciální zaměřovací matice a projekční matice je možné vymezit malou oblast, která se nachází poblíž např. kurzoru myši a tím vybrat konkrétní objekt scény.

Hlavním rozdílem mezi výběrem a zaměřením je interakce s uživatelem. Výběr je prováděn bez uživatelovy interakce, avšak zaměření se provádí kupříkladu až po kliknutí myši.

Pro lepší představu zde bude krátce popsán postup zaměření objektu. Mějme 2 objekty na obrazovce. Po kliknutí myši se zavolá procedura na vyhodnocení pozice kurzoru. Vrátí hodnotu a přepne aplikaci do módu výběru. Zde se inicializuje zásobník jmen a pronásobí matici zaměření aktuální ortografickou projekční maticí. Pro objekt, který je pod kurzorem myši se vyhodnotí zásah a prozkoumá se obsah zásobníku výběru, kvůli identifikaci objektu. Tímto postupem se identifikuje vybraný objekt.

Konkrétní ukázka kódu a detailnější popis výběru a zaměření je uveden v knize *OpenGL Průvodce programátora* [15, kapitola 14].

Výběr pomocí Ray Casting

Tato metoda výběru objektu se od předchozích dvou metod liší hlavně tím, že ji není nutné používat ruku v ruce s OpenGL. Ray Casting není primárně určen pouze pro výběr objektu ve scéně. Je to metoda, pomocí níž se řeší různé problémy v oblasti počítačové grafiky a výpočetní geometrie. Pojem Ray Casting se poprvé objevil roku 1982 v článku pana Rotha *Ray casting for modeling solids* [14].

[1] uint32 - 32 bitové bezznaménkové celé číslo (32 bit unsigned integer).

[2] Zásobník jmen - je místem pro navrácenou informaci o výběru objektů ve scéně. Je vytvořen při zápisu celočíselných jmen to něj, konkrétně v době vydávání příkazů kreslení v módu výběru.

Často může dojít k záměně výrazů Ray Casting a Ray Tracing. Proto je vhodné zde uvést rozdíl:

- Ray Casting

  Proces, kterým se analyticky počítají průsečíky, například promítnutého paprsku z kurzoru myši do scény, s nejbližším objektem ve scéně. V případě Ray Castingu bereme jako objekt například rovinu, kruh či jiná tělesa.

- Ray Tracing

  Komplexní výpočetně náročná technika generování realistických obrazů. Podrobnější popis Ray Tracingu obsahuje kniha *An Introduction to Ray Tracing* [3].

Že si jsou tyto dva pojmy blízké, dokazuje i kniha *Mathematics for 3D game programming and computer graphics* [6], kde je v kapitole 5 týkající se Ray Tracingu zmíněna možnost provést část výpočtu této metody pomocí Ray Castingu. Ray Casting se může v Ray Tracingu používat pro vyrenderování 3D scény.

3.2 Kolize

V reálném světě jsou všechny objekty hmotné, proto není problematické určit, zda-li se nějaké dva objekty dotýkají. V počítačovém světě to není tak samozřejmé. Je zde potřebné nastolit stejný řád, jaký panuje ve světě reálném. Nehmotným počítačovým objektům je nutné určit, jaké jsou jejich rozměry a umístění, aby nemohlo docházet k překryvům a přesahům mezi objekty.

Na detekci kolizí existuje velké množství algoritmů a jejich adaptací (viz online zdroj [1]). Každý je něčím unikátní, přesto se dají rozdělit na několik větších skupin, které přibližně shrnují jejich myšlenku. Lze je dělit na:

- algoritmy dělení prostoru,
- algoritmy ohraničení objektu,
- algoritmy průniku polygonů.

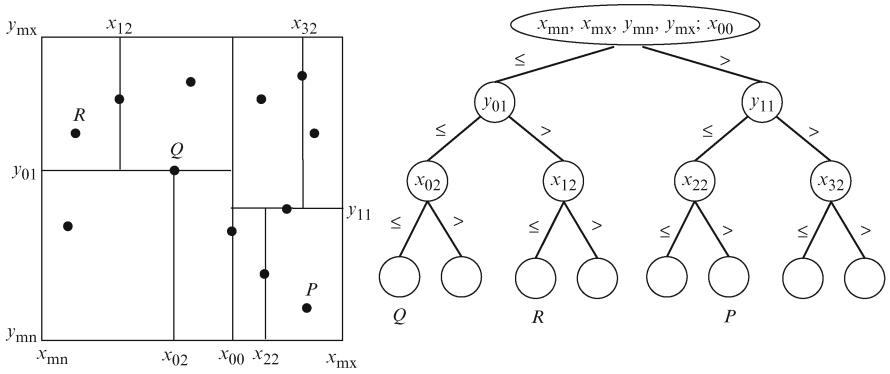
Algoritmy dělení prostoru

Pracují obecně v Eukleidovském prostoru[3]. Rozdělení tohoto prostoru vždy spočívá ve vytvoření dvou vzájemně disjunktních podprostorů. Tyto prostory se poté neovlivňují a mohou se na ně aplikovat další operace odděleně. Toto dělení umožňuje časově náročnou detekci kolize v celé scéně převést na detekci kolizí v části scény. Výhodou je menší složitost algoritmu a rychlejší výpočet. Pro zjištění, zda se některé objekty protínají nebo ne, je nutné vypočítat pouze zlomek z celkového objemu možností. Proto jsou tyto algoritmy využívány jako optimalizační prostředek při detekci kolizí.

[3] Eukleidovský prostor – v geometrii nejběžněji používaný prostor, který splňuje tzv. Eukleidovy postuláty (Eukleidovy axiomy) [12].

Datové struktury, které se při dělení prostoru používají jsou kategorizovatelné jako stromové struktury, například:

- BSP tree,
- quadtree,
- octree,
- k-d tree,
- bin,
- a další.



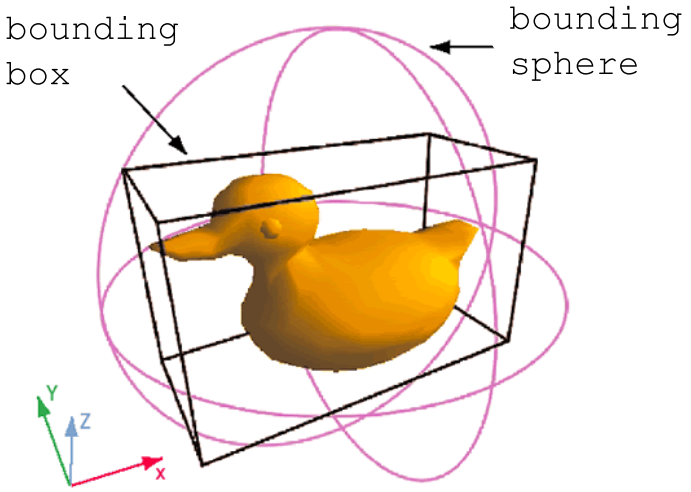
Obrázek 3.1: Znázornění principu algoritmů dělení prostoru. Příklad rozdělení prostoru do několika podprostorů a jejich následné převedení na stromový graf (http://what-when-how.com).

Algoritmy ohraničení objektu

Algoritmy ohraničení objektů patří mezi nejrychlejší, které se používají na detekce kolizí. Jejich princip spočívá v ohraničení každého objektu ve scéně neviditelným boxem, který má pravidelný tvar krychle, kvádru či koule. Pomocí boxů se potlačují speciální tvary komplexních objektů. Počítač poté vyhodnocuje, zda se protínají pouze tyto dodatečně vytvořené boxy.

V případě boxů ve tvaru kvádru či krychle, které využívá algoritmus s anglickým názvem Bounding boxes, se nejlépe pracuje, pokud jsou zarovnány podle souřadných os. Pokud tomu tak není, je nutné použít dodatečné algoritmy, které dokáží všechny boxy přepočítat do vzájemně porovnatelných boxů. Tento přístup není nutný při použití algoritmu Bounding spheres, kde jsou porovnávány boxy ve tvaru koule.

Oba dva zmíněné algoritmy jsou velmi rychlé, ale někdy nemusí dosahovat jejich aproximace reálného vzhledu objektu dostatečně přesných rozměrů, které mohou být u detekce kolize vyžadovány.

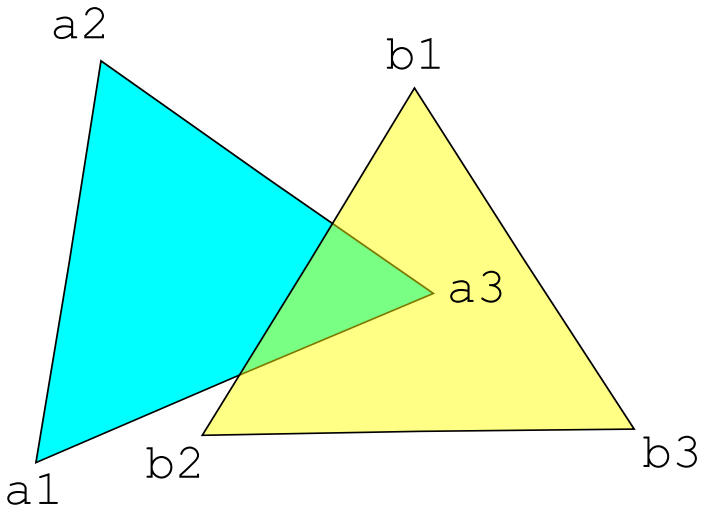


Obrázek 3.2: Princip algoritmů pro ohraničení objektu. Znázornění metody Bounding box a Bounding sphere (http://www.gamasutra.com).

Algoritmy průniku polygonů

Poskytují dostatečně přesnou detekci pro všechny konvexní[4] objekty. Principem je kontrola každého polygonu prvního objektu s každým polygonem druhého objektu. Pro kontrolu všech objektů ve scéně je tento přístup výpočetně náročný. Proto se kombinuje například s algoritmy ohraničení objektu. Po nalezení dvou kolidujících objektů se detekuje přesné místo, kde dochází k jejich průniku.

Tyto algoritmy lze použít i pro konkávní[5] objekty. Avšak než lze provést detekci kolize, je nezbytné konkávní objekty převést na konečnou množinu objektů konvexních.



Obrázek 3.3: Znázoznění algoritmu průniku konvexních polygonů.

[4]Konvexní – je označení objektu, který má všechny své stěny vypouké, vyklenuté směrem ven.
[5]Konkávní – opak konvexních objektů; jsou to objekty vyduté směrem dovnitř.

Kapitola 4

Návrh aplikace

Prvotní fází každého vývoje je návrh. Jeho kvalita se odvíjí zpravidla od zkušeností návrháře. Důležité je pokrýt všechny nezbytné části budoucí aplikace bez konkrétních implementačních detailů. Ty jsou v této fázi nepotřebné. Základem je stanovení několika požadavků, které jsou postupem času rozvíjeny a konkretizovány.

Hlavním bodem pro samotný vznik je výběr jednotlivých technologií a postupů, jak bude výsledná aplikace vytvářena. Dnes již není potřebné, aby vývojář vyvíjel všechny funkcionality aplikace sám. Existuje mnoho volně šiřitelných knihoven a technologií, které potřebné funkcionality poskytují.

4.1 Požadavky na aplikaci

Cílem této práce je návrh a implementace interaktivní stavebnice pomocí OpenGL. Aplikace by měla poskytnout uživateli prostředí pro tvorbu modelů s využitím LEGO® kostek. Je navržena tak, aby byla rozšiřitelná o další modely kostek, které v ní bude možné využít. Sestavována bude pomocí CMake[1].

Hlavními dvěma prvky, do kterých lze zahrnout vše ostatní, jsou menu a herní plocha. Po spuštění aplikace má uživatel k dispozici čistou herní plochu s výchozím nastavením menu. Tím se rozumí, že je aplikace spouštěna vždy česky s otevřenými panely s ovládacími a informačními prvky. Panely lze libovolně přemisťovat po herní ploše. Je možné je vypnout či zapnout pomocí menu.

[1] CMake – viz https://cmake.org/

Menu

Hlavní menu aplikace se nachází v horní liště, která je rozdělená na několik záložek. Každá z nich poskytuje několik možných interakcí s aplikací:

- Soubor - Nový, Načíst, Uložit, Uložit jako, Exportovat seznam kostek, Ukončit,
- Upravit - Zpět, Vpřed, Smazat označenou kostku, Smazat všechny kostky,
- Nástroje - Panel nástrojů (manipulace), Panel kostek (výběr z typů), Panel počtu kostek (informace o scéně),
- Nápověda - Ovládání klávesnice, Ovládání myši,
- O aplikaci - Jazyk (Anglicky, Česky), Verze.

Načtení scény ze souboru a uložení celé scény do souboru je možné na jakékoliv místo v počítači, které si uživatel zvolí. Ukládané a načítané soubory mají speciální příponu .bgl.

Herní plocha

Je vyobrazena pomocí nejběžnější barvy pro podkladové desky u stavebnice LEGO®, tzn. tmavě zeleně. Uživatel má možnost se po herní ploše pohybovat pomocí šipek dopředu, dozadu i do stran. Má možnost natáčet kameru pohledu i oddalovat a přibližovat scénu. Díky tomu lze „procházet“ mezi jednotlivými modely vytvořenými ve scéně. Velikost herní plochy je pevně nastavena na rozměr 64×64. Ten byl vybrán jako dvojnásobný rozměr standartní podložky na stavbu lega. Původní rozměr je 32×32, avšak pro herní plochu aplikace byl poměrně malý. Kvůli možnosti stavět větší modely se zdá být tento rozměr dostačující.

4.2 Modely v aplikaci

Modely se dají získat několika možnými způsoby, které jsou zmíněny v podkapitole 2.1. V této práci je využitý přístup podrobněji rozvedený ve zmíněné podkapitole v části Modely tvořené člověkem. Proto bylo potřebné najít technologii, která umožní nahrání modelu ze souboru do aplikace. Zvolena byla Open source knihovna Assimp[2]. Ta podporuje všechny běžně dostupné formáty 3D modelů.

Modely LEGO® kostek se na internetu dají najít nejčastěji ve formátech OBJ nebo STL. Rozíl v těchto formátech spočívá v pojetí objektu samotného. OBJ reprezentuje geometrii 3D objektu. Zahrnuje informace o poloze každého vrcholu, UV pozici každého vrcholu (slouží pro určení textury) a normálách vrcholů i povrchů. Povrchy mají každý polygon definovaný pomocí seznamu vrcholů a texturovacích vrcholů (viz [8]).

Oproti tomu STL formát popisuje pouze tvar 3D objektu a zanedbává informace o barvě, textuře a dalších možných atributech. Často mívá binární reprezentaci (viz [5]).

Pro návrh této aplikace byl použit formát STL, neboť některé atributy formátu OBJ by nebyly využity. Například informace o barvě není potřebná díky možnosti přebarvení kostek v aplikaci.

[2] Assimp - viz www.assimp.org

4.3 Reprezentace scény

Struktury, které pro reprezentaci scény interaktivní stavebnice přicházely v úvahu, byly graf scény (viz podkapitola 2.2) a zobrazovací seznam. Tyto dvě struktury se liší náročností implementace a tím, pro které aplikace se která z nich více hodí. Společné mají využití. Obě struktury slouží k zapamatování všech vykreslovaných objektů a jejich atributů ve scéně.

Graf scény se hodí pro rozsáhlé herní scény. Takové mohou obsahovat například počítačové hry RPG žánru, kde se vyskytuje velké množství sobě podobných objektů, ať už tvarem, strukturou či barvou.

Pro malé aplikace, jakou je interaktivní stavebnice, je graf scény zbytečně komplikovaný a náročný na implementaci. Tento typ si vystačí se zobrazovacím seznamem. Ten obsahuje informace o každé kostce scény a uchovává její atributy. Oproti grafu scény je jednodušší na implementaci.

Scéna interaktivní stavebnice je rozvržena do mřížky, díky které je zobrazování snazší. Polohy kostek mohou být měněny pouze po 1 bodu (platí pro všechny tři souřadné osy), což věrně kopíruje chování reálné LEGO® stavebnice. Navíc celočíselná aritmetika je v počítači výpočetně rychlejší.

4.4 Vykreslování herní plochy

Pro vykreslování herní plochy, včetně kostek na ní umístěných, je důležité mít uložené všechny potřebné informace o každé kostce v zobrazovacím seznamu. Kostka je zde reprezentována svým typem, barvou, pozicí, rotací a informací o tom, zda-li je vykreslena nebo skryta. Pro správné vykreslení jsou potřebné všechny zmíněné informace. Pokud by některé z nich nebyly správně zapamatovány, mohlo by dojít k nepřesnostem či selhání aplikace.

Podle zobrazovacího seznamu scény (viz podkapitola 4.3) se nastavují vstupní proměnné OpenGL, pomocí kterých je vykreslena výsledná podoba aplikace.

4.5 Řešení výběru kostek

V návrhu aplikace se počítalo s využitím i na počítačích, kde myš není běžnou součástí. Proto lze základní funkcionality aplikace ovládat čistě pomocí klávesnice. Využití obou prostředků je optimální cestou, díky které lze využít všechny nabízené funkce. Manipulace s aplikací striktně jedním z prostředků je pro uživatele nepohodlná a značně omezující. Mezi zařízení bez myši (myšleno v základním vybavení) lze zařadit například notebooky.

Určení konkrétní kostky pro výběr je pomocí klávesnice navrženo jako průchod seznamem. Po stisku klávesy se přepne aktuálně vybraná kostka na další, za ní v seznamu následující, kostku. Pro modely, které budou obsahovat více než cca 20 kostek je tento způsob práce nevhodný.

Výběr objektu pomocí myši je pro uživatele pohodlnější variantou a dá se řešit několika způsoby. Zmíněné jsou v podkapitole 3.1. Zde byl použit způsob výběru pomocí Ray Castingu (viz podkapitola 3.1, část Ray Casting). Díky jednotkovému mřížkovému rozvržení herní scény se zde ukázalo řešení pomocí Ray Castingu jako vhodné. Výpočet se díky pravidelnosti mřížky značně zjednodušil.

4.6 Detekce umístitelnosti kostek

Jak již bylo zmíněno v podkapitole 3.2, existují tři kategorie algoritmů, které se zabývají detekcí kolizí. Zarovnání celé zobrazované scény do mřížky usnadnilo výběr algoritmů. Jako vhodný kandidát se zde ukázala skupina algoritmů ohraničení objektu (viz podkapitola 3.2, část Algoritmy ohraničení objektu).

V aplikaci se využívá principu ohraničení objektu, avšak není to ohraničení v pravém slova smyslu. Není zde potřebné vytvářet nové boxy kolem již existujících objektů. Všechny kostky v aplikaci jsou zasazeny do mřížky. Nejmenší část této mřížky je krychle s rozměry $1 \times 1 \times 1$ a sama sobě je ohraničujícím boxem. Tyto části nelze dále dělit a každá kostka se skládá z několika takových částí. Proto zde byla za strukturu uložení obsazené části zvolena mapa, která umožňuje pokrýt skoro nekonečnou scénu, aniž by byl zvýšen nárok na paměťové zdroje. Klíčem mapy jsou souřadnice x, y, z, které určují konkrétní pozici kostky. Následně je hodnota (v tomto případě záznam obsahující identifikátor kostky) namapována na konkrétní klíč. Pozice v mapě je volná, pokud v ní není uložen záznam.

Aby se určilo, zda může být kostka umístěna na uživatelem vybrané místo, je nutné projít mapu a porovnat, zda jsou nově obsazované pozice stále volné. Zároveň se kontroluje, že aspoň jedna z okolních pozic je v mapě obsazena. Pokud by žádná obsazena nebyla, není možné umístit kostku na vybranou pozici, kvůli fyzice modelu. Kostka na svém místě nemůže držet, pokud nebude nad nebo pod ní aspoň jedna část kostky jiné. Tzn. kostka nemůže být připojena ke zbytku modelu bez opory.

4.7 Uložení a načtení modelované scény

Modely, které uživatel na herní desce vytvoří, je důležité uložit, aby si je mohl později opět načíst. Proto byl vytvořen formát .bgl, který obsahuje celočíselné informace o kostkách, které se nacházejí na jedné herní ploše. Výhodou tohoto úzce specializovaného formátu uložení herní scény je jeho přehlednost. Navíc soubory tohoto typu jsou textové a nedosahují velkých velikostí. Uložený soubor uchovává údaje o typu, umístění a barvě každé kostky z herní plochy. Každá kostka je uložena na jednom řádku.

Díky zmíněnému členění je možné se v souboru rychle zorientovat a hledání atributů kostky je jednodušší. Pro načtení herní scény se uloží obsah jednotlivých řádků a sloupců do příslušných proměnných a poté se vykreslí celá načtená scéna.

4.8 Export seznamu kostek

Aplikace umožňuje export souboru s počty a barvami potřebných kostek pro vytvoření reálného modelu. Nejvhodnější formát je .txt, aby ho bylo možné otevřít na jakémkoliv počítači. Výsledkem je tedy textový soubor s nezbytnými údaji o použitých kostkách. Informace dostupné v tomto souboru jsou potřebné pro možnost objednání přesného počtu kostek konkrétních typů a barev.

4.9 Ergonomie

Ergonimie je v herním světě velmi důležitou veličinou. U množství recenzí je patrné, že hry, které mají kvalitně navrženou ergonomii, což může být i na úkor grafických prvků nebo dějové linie, jsou výrazně lépe hodnoceny. Důvodem je fakt, že více reflektují hráčské pohodlí. Na druhé straně stojí hry s propracovaným dějem i grafikou, ale bez správně vyladěného ovládání. Tato kategorie může přijít o potencionální hráče právě kvůli podcenění významu ergonomie. Avšak hodnocení her jsou subjektivní a nedají se globalizovat.

V každé hře, jenž by měla být interaktivní, je potřebné se ergonomií zaobírat. Již několik generací počítačů je zřejmé, že klávesnice je dobrým základem každé hry, ale aby byla hra opravdu poutavá, je potřebné, aby bylo ovládání umožněno i myší.

Do té doby, než byly k dispozici myši, byl počítač záležitostí výhradně pro nadšence. Jak se historicky prokázalo, od připojení první myši a vzniku grafického uživatelského rozhraní, poptávka po počítačích, nejen jako velkých a drahých výpočetních strojích, mohonásobně vzrostla.

Myš je ergonomičtější než klávesnice, proto si na ni lidé rychle navykli. Všechny moderně zpracované hry kombinují obě zařízení a získávají si tak lepší ovladatelnost a uživatelskou přístupnost.

Jakákoliv hra, která se odehrává v 3D prostoru, by měla umožnit ovládání oběma prostředky. Navrhováním ergonomie v herním světě se zabývají vývojářská studia, kde počítačové hry vznikají. Materiály k tomuto tématu se na internetu nedají najít, protože spadají do know-how jednotlivých vývojářských firem.

Kapitola 5

Implementace

Po uskutečnění návrhu aplikace, který je rozebírán v kapitole 4, je dalším krokem implementace. V této fázi je potřebné mít již vybrané technologie, pomocí nichž bude aplikace realizována. Ty ovlivňují výběr knihoven, které jsou využity pro zjednodušení implementace jednotlivých funkcionalit.

5.1 Technologie

Hlavním požadavkem na technologie je výběr jazyka C++ a OpenGL. Jazyk C++ je vhodný kvůli možnosti objektového návrhu. OpenGL je výhodné díky grafické akceleraci, což znamená, že výpočty prováděné při vykreslování jsou počítány na grafické kartě. Tyto dvě technologie jsou vzájemně napojitelné a jejich kombinace je dobrým základem pro vznik aplikace. Konkrétně jsou použity C++ 11 a OpenGL 4.0.

5.2 Knihovny a nástroje třetích stran

Základem aplikace je OpenGL[1]. Tato grafická knihovna zaštiťuje základní zobrazovací techniky aplikace. Na pokrytí funkcionalit bylo potřebné rozšířit čisté OpenGl o jeho nádstavby jako jsou GLEW[2] a GLM[3]. Knihovna SDL2[4] je využita na práci se vstupními zařízeními, jako je klávesnice a myš, aby bylo možné zachytávat jimi vyvolané události. Slouží také k prvotní inicializaci okna aplikace.

Pro načítání modelů kostek je použita knihovna Assimp[5]. Umožňuje nahrávat soubory s již vytvořenými 3D modely. Změnu typu nově vkládané kostky nebo změnu typu kostky, která je již v aplikaci je možné navodit u některých vybraných modelů pomocí klávesnice. Dostupnost veškerých modelů nahraných v aplikaci je zprostředkována pomocí Panelu kostek (viz 4.1). V tomto panelu jsou načteny obrázky jednotlivých kostek pomocí knihovny FreeImage[6], která umožňuje nahrání několika formátů obrázků, jako jsou PNG, BMP, JPEG, TIFF a další.

[1] OpenGL - viz https://www.opengl.org/.
[2] GLEW - viz http://glew.sourceforge.net/.
[3] GLM - viz http://glm.g-truc.net/0.9.8/index.html.
[4] SDL2 - viz https://www.libsdl.org/.
[5] Assimp - viz http://assimp.sourceforge.net/.
[6] FreeImage - viz http://freeimage.sourceforge.net/.

Menu v záhlaví a panely aplikace jsou vytrořeny za podpory knihovny ImGUI[7]. Ta umožňuje realizaci uživatelského grafického rozhraní, které se dá přizpůsobit na míru aplikaci, aby zapadalo do celkového konceptu.

Načítání a ukládání souborů v aplikaci je implementováno pomocí knihovny Native File Dialog[8], která umožňuje načítat a ukládat soubory v jakémkoliv formátu. Umístění kam je ukládán či odkud je načítán soubor je čistě v kompetenci uživatele.

5.3 Atributy kostky

Každá kostka je reprezentována několika svými atrubity. Ty jsou potřebné pro správné zobrazení celé scény aplikace. Při uložení do souboru se zapíší atributy pro každou jednotlivou kostku na nový řádek. Každý atribut má svůj význam, bez kterého by byla scéna nesprávně vykreslena. Těmito atributy jsou:

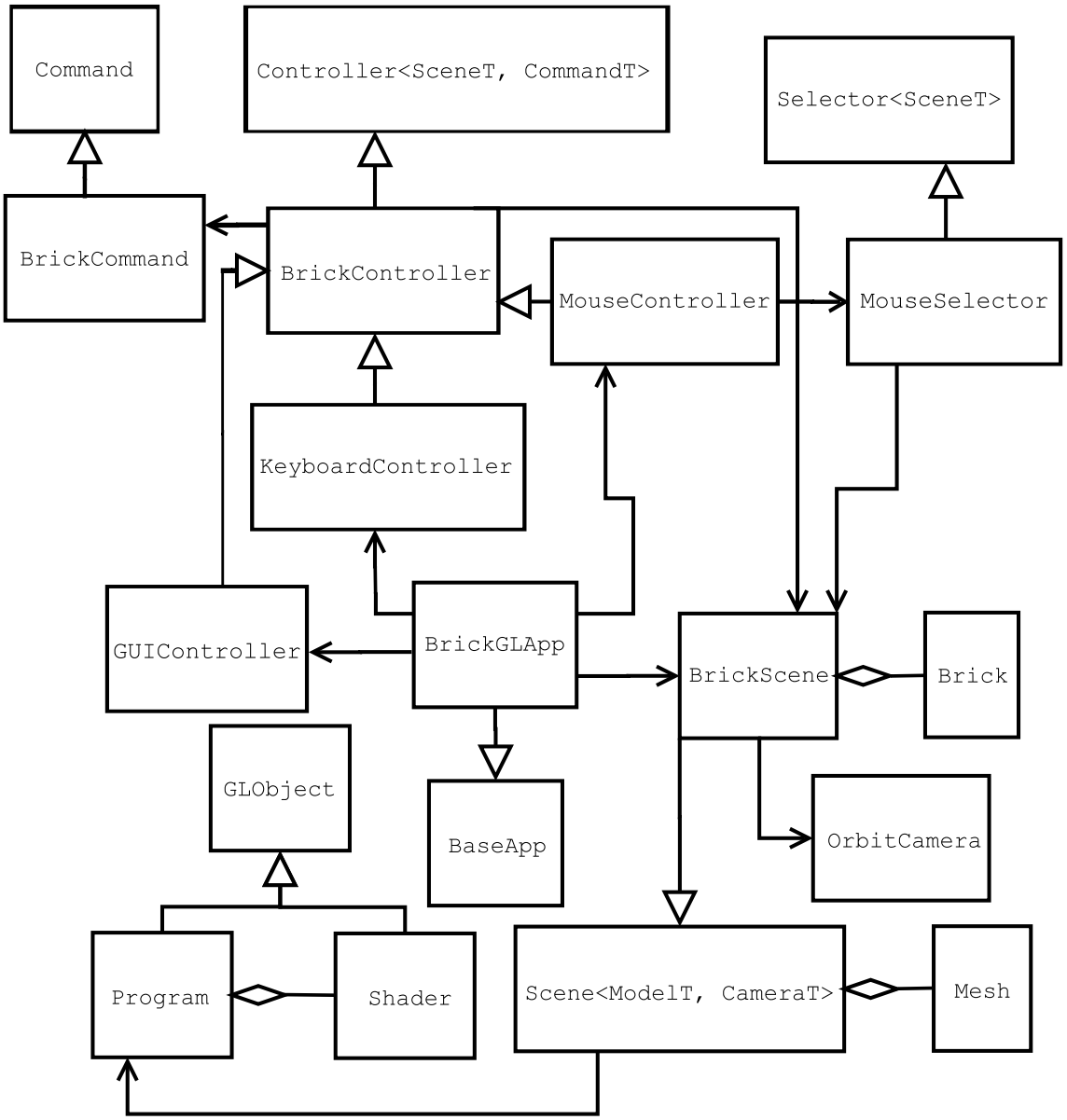
- `Rotation rotated` - natočení kostky ve scéně; kostka může být natočena o 0, 90, 180 nebo 270 stupňů,
- `bool hidden` - nabývá hodnoty true, pokud je kostka skryta; jiank je false,
- `glm::vec3 position` - poskytuje informaci o pozici kostky ve scéně; reprezentováno trojrozměrným vektorem (x, y, z),
- `glm::vec4 color` - určení barvy kostky; udáváno pomocí čtyřrozměrného vektoru (obsahuje složky RGB a úroveň alfa),
- `Type type` - říká jakého je daná kostka typu; na výběr je z několika typů kostek, které jsou k dispozici.

5.4 Důležité třídy aplikace

Jazyk C++ podporuje objektově orientované paradigma, které je v dnešní době kvůli jeho výhodám prosazováno. Tímto přístupem se dá dosáhnout dostatečné míry abstrakce, aby bylo možné jednotlivé třídy objektů použít bez problémů i mimo stávající projekt. Znovupoužitelnost spočívá v zapouzdření metod ke konkrétnímu typu objektu.

Tato aplikace je implementována podle objektově orientovaného návrhu. Na obrázku 5.1 je patrné hiearchické uspořádání a návaznosti jednotlivých tříd.

[7] ImGUI - viz https://github.com/ocornut/imgui.
[8] Nafive File Dialog - viz https://github.com/mlabbe/nativefiledialog.



Obrázek 5.1: Hiearchické uspořádání tříd v rámci aplikace.

Controller

Všechny třídy s přídavkem `Controller` dědí od hlavní `Controller<SceneT>` třídy. Obsahují logiku ovládání herní scény. Zacyhtávají události ze vstupních zařízení. Na základě příchozí události provedou příslušnou akci.

Každý `Controller` zajišťuje obsluhu jemu náležících vstupních zařízení a částí aplikace. `BrickController` specializuje `Controller` na práci s kostkami. `MouseController` zachytává události vyvolané pohyby myši. `KeyboardController` se zaměřuje na události vyvolané stiskem klávesnice. `GUIController` bere na zřetel události obou vstupních zařízení a nastavuje podle nich grafické uživatelské rozhraní aplikace.

Command

`Command` třída se vztahuje k návrhovému vzoru Command[9]. Slouží k zajištění undo a redo operací aplikace. Třída `Command` reprezentuje jednu operaci, která je uložená v třídě `Controller`. `BrickCommand` třída se specializuje na operace s kostkami.

Selector

Třídy `Selector` obsahují logiku zajišťující výběr kostek pomocí myši. `MouseSelector` obsahuje metodu na vybrání kostky pod kurzorem myši. Tyto třídy jsou nezbytnou součástí interaktivní stavebnice, protože zaměření objektů pomocí myši patří mezi základní požadavky, aby byla aplikace komfortní pro uživatele.

Scene

`Scene` třídy uchovávají informace o scéně. Hlavní z nich je `Scene<ModelT, ObjectT>`, od které dědí `BrickScene`. Hlavní třída schraňuje informace o jednotlivých modelech kostek. Ty jsou instancemi třídy `Mesh`. Následně jsou skrze třídu `Scene<ModelT, ObjectT>` dostupné v celé aplikaci.

Oděděná třída `BrickScene` je konkretizací nadřazené třídy. Uchovává v sobě informace o jendotlivých kostkách scény, jimiž jsou instance třídy `Brick`. Každá kostka obsahuje mimo jiné informaci o svém typu, který určuje, jaká instance třídy `Mesh` bude vykreslena.

`OrbitCamera` je úzce spjata se scénou. Obsahuje nastavení kamery (zoom, rotace, pozice) a zajišťuje výpočty transformačních matic potřebných pro korektní vykreslení scény.

App třídy

`BaseApp` a od ní oděděná `BrickGLApp` jsou hlavními třídami celé aplikace. `BrickGLApp` slouží jako vstupní bod. Zde se promítne výchozí nastavení (uložené v hlavičkovém souboru `Configuration`) do celé aplikace. Z této třídy je volán nekonečný cyklus `draw`, který v pravidelných intervalech překresluje celou scénu, aby odpovídala aktuálnímu uživatelskému nastavení a obsahu scény.

Třídy pro práci s OpenGL

V hiearchii jsou zahrnuty tři třídy, které slouží pro práci s OpenGL. První je `GLObject`. Má na starosti zapouzdření `GLuint` proměnné, která definuje ID OpenGL objektu. V této triádě je hlavní třídou, od které zbylé dvě dědí. Druhou třídou je `Shader`. Její funkcionalitou je načítání shaderů do aplikace ze souboru či vstupního řetězce. Poslední třídou je `Program`. Ta zajišťuje linkování shaderů do jednoho programu. Ten je použit pro vykreslování scény.

[9]Command - část principu převzata z http://www.oodesign.com/command-pattern.html.

5.5 Problémy vzniklé při implementaci

I dobrý návrh se může potýkat s nedostatky. Míra těchto nedostatků se úměrně odvíjí od zkušenosti návrháře s konkrétním typem aplikace a využitými technologiemi. Proto se často stává, že až v průběhu implementace vyvstanou problémy, se kterými se při návrhu nepočítalo. Tato aplikace není výjimkou. I zde nastaly problémy, které musely být v průběhu implementace vyřešeny.

Podpora češtiny

Tento problém se vyskytl při zapracování řetězců v grafickém uživatelském rozhraní (GUI). C++ 11 již umožňuje práci s UTF-8[10] řetězci, avšak knihovna ImGUI (viz 5.2) má zabudovanou podporu pouze pro některé jazyky. Proto bylo nutné vyhledat Unicode rozsah vymezený pro češtinu a ten předat fontu grafického rozhraní. Zároveň bylo potřebné vybrat font, který podporuje znaky češtiny. Byl zvolen font GFSNeohellenic[11].

Detekce kolizí

Problém detekce kolizí se dá řešit několika způsoby. Jsou popsané v kapitole 3.2. I když v případě této aplikace jsou objekty zjednodušeny na kvádry, bylo stále dosti náročné tuto detekci zajistit. Výsledná implementace se opírá o mřížkové rozdělení herního světa a pravidelné tvary objektů v aplikaci. Pokud by byla rozšířena o modely kostek, který by nebylo možné reprezentovat pomocí kvádrů, bude potřebné implementovaný způsob kolizí upravit.

Zobrazení vybrané kostky

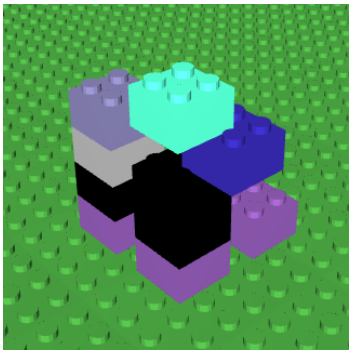
Pro lepší přehlednost aplikace je nutné, aby byla vybraná kostka odlišena od ostatních kostek ve scéně. Odlišení bylo navrženo a implementováno třemi způsoby. Prvním implementovaným způsobem bylo zesvětlení kostky oproti její původní barvě (viz obrázek 5.2). Tento přístup byl jednoduše realizovatelný a nevytěžoval velké množství zdrojů. Ukázal se být nevhodný, neboť způsoboval zkreslení výsledné barvy. Při změně barvy kostky (přičemž musí být kostka vybraná) nebylo patrné, jak bude po umístění (odznačení kostky) vypadat.

Druhým odzkoušeným způsobem byla průhlednost (viz obrázek 5.3). Jevila se jako vhodný kandidát, neboť nezkresluje výslednou barvu v takové míře, jako první přístup. Dalším bonusem byla atraktivní vizuální stránka takto vykreslených kostek. Hlavním negativem a také důvodem, proč byl tento způsob zavržen, byla vysoká náročnost na vykreslování. Rozdíl mezi prvním a druhým způsobem bylo několik desítek vykreslených snímků za vteřinu. To bylo pro plynulost aplikace znatelné a bylo nutné od tohoto přístupu upustit.

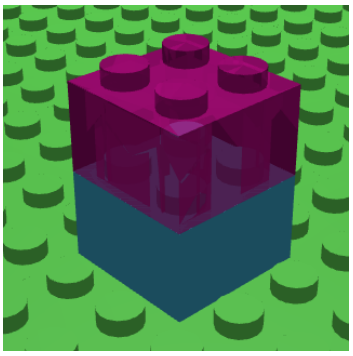
Poslední a zároveň vítězný implementovaný způsob je zvýraznění kostky pomocí animace (viz obrázek 5.4). Jako způsob odlišení kostky se nabízela jako nejlepší kandidát. Nejen, že animace jsou obecně uživatelsky přitažlivé, ale navíc nevytěžují neúměrně zdroje. Počet vykreslených snímků za vteřinu odpovídal prvnímu způsobu.

[10] UTF - Unicode Transformation Format.

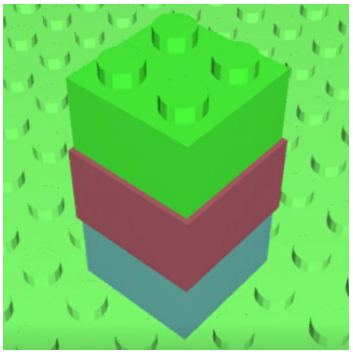
[11] GFSNeohellenic - získán z http://www.ceskefonty.cz.



Obrázek 5.2: Metoda zobrazení vybrané kostky pomocí změny barvy. Vybraná azurová kostka nelze rozlišit.



Obrázek 5.3: Metoda zobrazení vybrané kostky pomocí průhlednosti. Poloprůhledná fialová kostka je dobře viditelná.



Obrázek 5.4: Metoda zobrazení vybrané kostky pomocí animace. Na obrázku není animace červené kostky příliš patrná.

Kapitola 6

Výsledky měření

Po dokončení implementace je vhodné demonstrovat výslednou aplikaci pomocí metrik. Těmi mohou být například závislosti jejího výkonu nebo spotřebovaných zdrojů na jejím zatížení. Následující podkapitola se zabývá výsledky měření, které jsou znázorněny pomocí grafů.

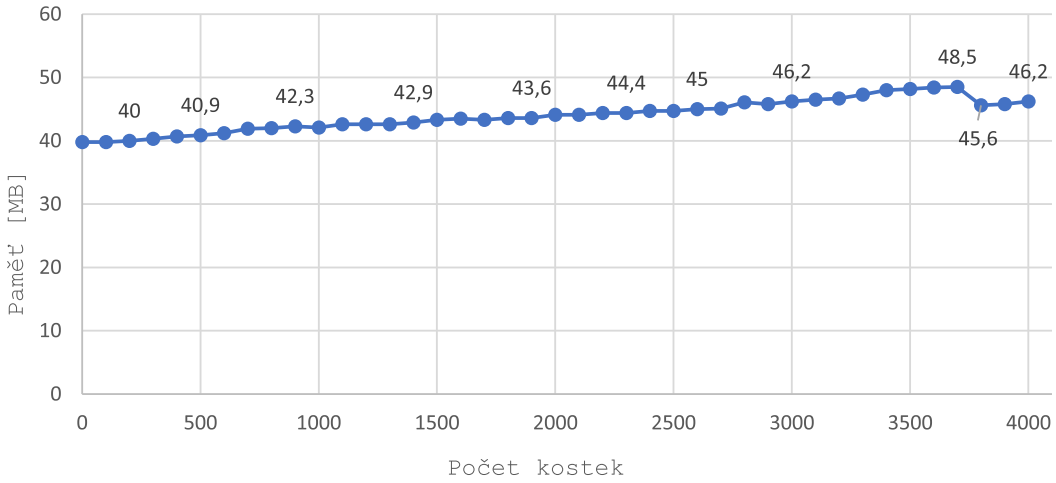
Výpovědní hodnotu o naimplementované aplikaci zde poskytují grafy. Každý graf musí mít svou výpovědní hodnotu a účel za kterým je pořizován. Charakteristiky zpracované pomocí grafu bývají dobrými ukazateli vhodnosti použitých technik při implementaci aplikace. V této podkapitole jsou uvedeny grafy, které se zabývají různými aspekty aplikace v závislosti na počtu kostek ve scéně.

Všechny údaje byly měřeny na stejném referenčním stroji Lenovo B590: Intel Core i3 (2nd Gen) 2328M / 2.2 GHz; Dual-Core; DDR3 SDRAM 8GB 1333 MHz; Intel HD Graphics 3000. Hodnoty grafů byly získány za stejných podmínek, tzn. po přidání přesně 100 kostek, po ustálení měřených hodnot.

Metodikou získávání hodnot pro grafy 6.1 a 6.3 bylo využití programu Správce úloh. Hodnoty grafu 6.2 byly získány pomocí konzolových výpisů rychlosti vykreslení jednoho snímku aplikace.

Graf 6.1 zobrazuje závislost nárůstu paměti v MB na počtu kostek ve scéně. Je zde patrné, že se zvyšujícím se počtem kostek roste i míra alokované paměti. Roste téměř lineárně, proto se dá předpokládat, že bude mít přímo úměrné rostoucí tendence i pro více kostek ve scéně, než je v grafu zaneseno.

Při stejném měření se paměť alokovaná pro grafickou kartu pohybovala kolem 60 MB. Míra využité paměti nebyla vysoká, neboť každý model kostky je v grafické paměti uložen pouze jednou.



Obrázek 6.1: Graf závislosti nárůstu paměti na počtu kostek ve scéně aplikace.

Doba potřebná pro vykreslení jednoho snímku v závislosti na počtu kostek ve scéně je zobrazena grafem 6.2. Hodnota je udána v milisekundách. Tendence zobrazeného grafu je s přibývajícím počtem kostek rostoucí. Z grafu je patrná přímá úměra obou měřených veličin.

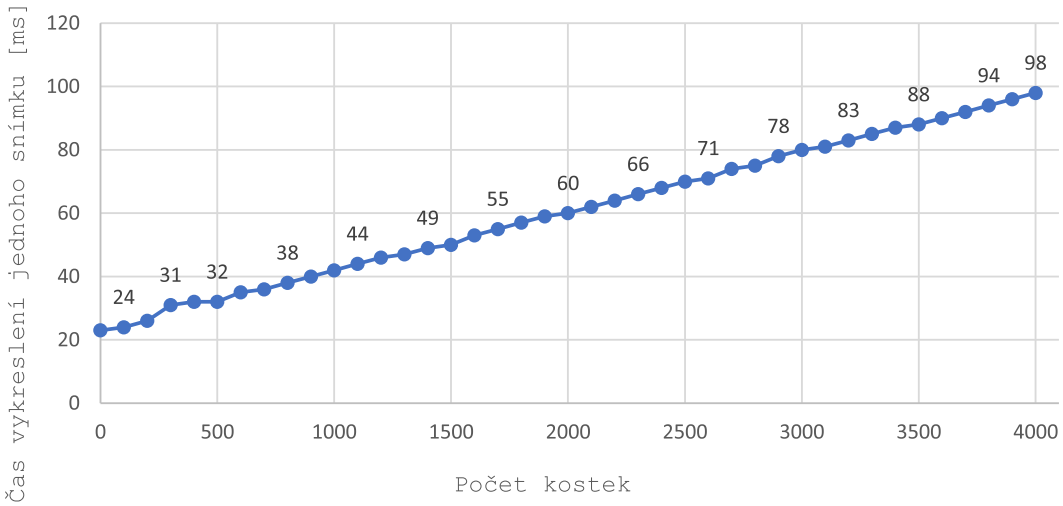
Při prázdné scéně je hodnota počtu vykreslených snímků za sekundu[1] přibližně 44. Prázdná scéna obsahuje pouze podložku, která je složena z 256 kostek jednoho typu. Kolem 1000 kostek klesne FPS na 25. Hranice mezi 25 až 30 FPS je brána jako minimum pro plynulou grafiku. Moderní hry se pohybují nad 60 FPS, což je obvyklá snímkovací frekvence lidského oka. Scénu, vykreslovanou nižší snímkovací frekvencí, nevnímá lidské oko plynule.

Poslední měření, které je znázorněné grafem 6.3, udává závislost zátěže procesoru (CPU) udanou v procentech na počtu kostek ve scéně. Vytíženost procesoru je zde zapříčiněna přeposíláním dat grafické kartě, která je zpracovává a přijaté údaje vykresluje na obrazovku počítače.

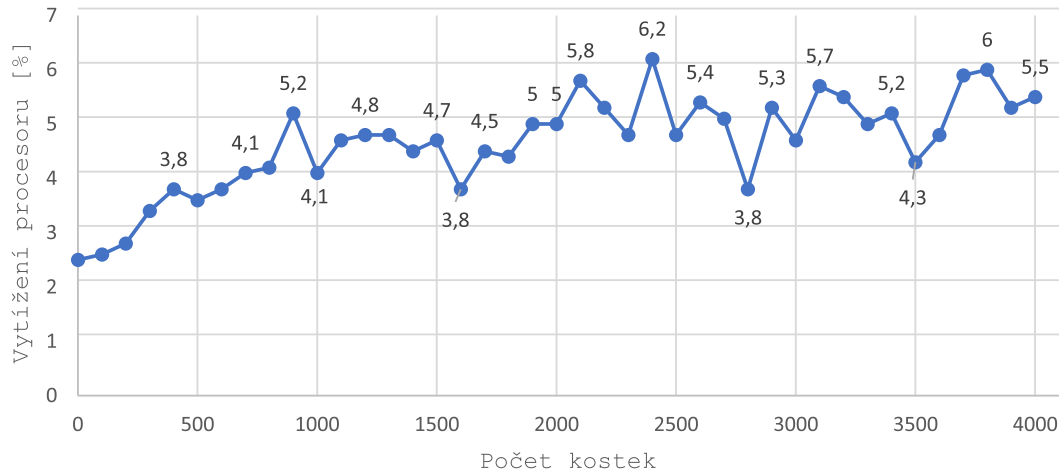
Vytížení procesoru roste velmi pozvolna v závislosti na přibývajícím počtu kostek. V grafu jsou patrné značné výkyvy, které jsou pravděpodobně způsobeny různou lokalitou odkazů[2] v datové struktuře pro uložení scény.

[1] Počet vykreslených snímků za sekundu - nejběžněji se setkáváme s anglickou zkratkou FPS (Frames per second).

[2] Lokalita odkazů - míra toho, kolik různých shluků adres bude proces potřebovat v krátkém časovém úseku. Definice převzata z přenášky [16].



Obrázek 6.2: Graf závislosti času potřebného na vykreslení jednoho snímku na počtu kostek ve scéně aplikace.



Obrázek 6.3: Graf závislosti nárůstu zátěže procesoru (CPU) na počtu kostek ve scéně aplikace.

Kapitola 7

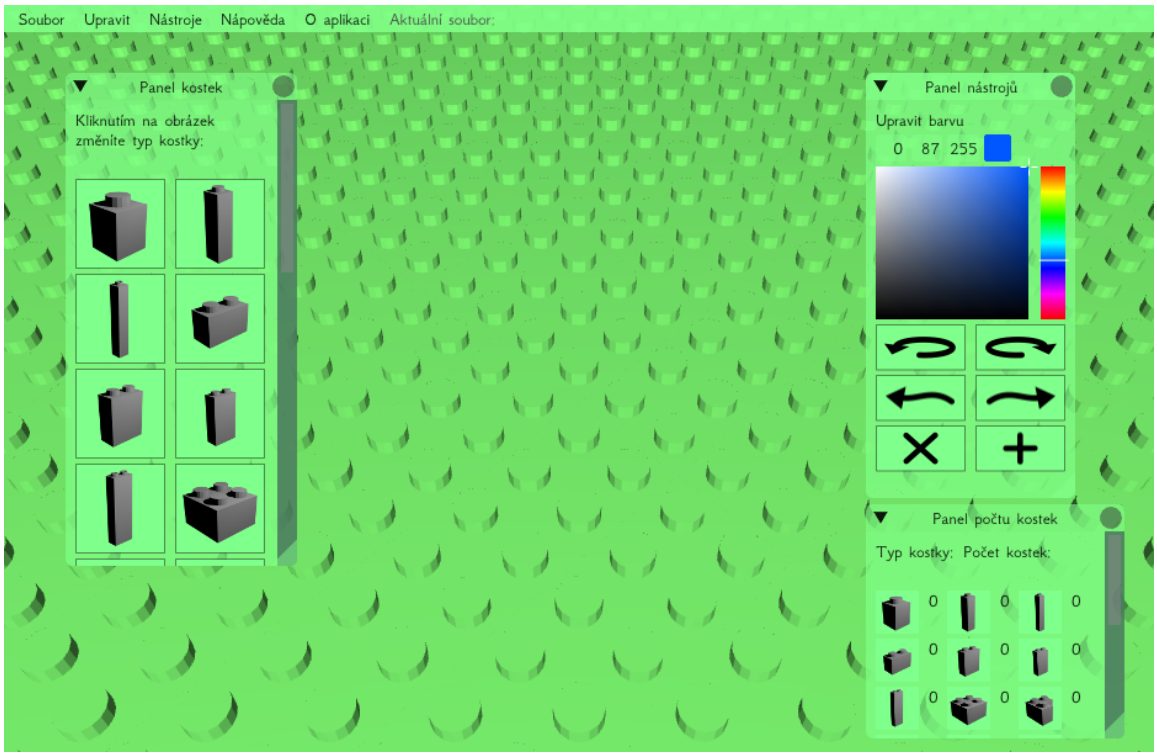
Závěr

Výsledkem práce je interaktivní stavebnice typu LEGO® implementovaná pomocí C++, OpenGL a několika dalších knihoven, které jsou zmíněny v kapitole 5.2. Aplikace nabízí možnost výběru jazyka mezi češtinou a angličtinou. Panely pro práci s kostkami jsou přemístitelné, je možné si přizpůsobit obrazovku dle vlastních preferencí. Nabídka menu byla navržena s ohledem na zkušenosti s aplikacemi podobného typu, jakými jsou například grafické editory.

Aplikace je do budoucna rozšiřitelná o další modely kostek. Ty lze přidávat v podobě člověkem vytvořených modelů nebo generovat procedurálně. Tím by se pokryla většina pravidelných tvarů. Některé pro LEGO® specifické dílky, jakými jsou různé spoje, kola a další, by bylo možné do aplikace také začlenit. To by ovšem vyžadovalo jiné rozdělení herního prostoru aplikace a složitější logiku práce s modely.

Pro zrychlení aplikace a zmenšení výpočetní náročnosti by bylo možné optimalizovat vykreslování kostek pouze na ty, které jsou v jednom okamžiku ve scéně viditelné. Tato optimalizace je vhodná především pro modely, které budou čítat více než 1000 kostek, což je patrné z grafu 6.2.

Návrh a implementace aplikace vyžadovaly rozšíření znalostí práce s jazykem C++, nastudování OpenGL, jeho nádstaveb a dalších pro aplikaci nezbytných knihoven. Na obrázku 7.1 je zobrazena výsledná aplikace BrickGL ve výchozím nastavení.



Obrázek 7.1: Výchozí stav aplikace BrickGL.

Literatura

[1] Bake, M. J.: *Euclidean space*. [online]. [cit. 2017-03-22].
URL http://www.euclideanspace.com/threed/animation/collisiondetect/

[2] Cohen, M. F.; Wallace, J. R.: *Radiosity and Realistic Image Synthesis*. Academic Press, 1993, ISBN 978-0-12-178270-2, 381 s.

[3] Glassner, A.: *An Introduction to Ray Tracing*. The Morgan Kaufmann Series in Computer Graphics, Elsevier Science, 1989, ISBN 978-0-080-499055.

[4] Gouraud, H.: *Continuous Shading of Curved Surfaces. IEEE Trans. Comput.*, ročník 20, č. 6, Červen 1971: s. 623–629, ISSN 0018-9340, doi:10.1109/T-C.1971.223313, [online]. [cit. 2017-03-17].
URL http://page.mi.fu-berlin.de/block/htw-lehre/wise2015_2016/bel_und_rend/skripte/gouraud1971.pdf

[5] Hull, C.; Lewis, C.: *Stereolithographic supports*. Listopad 2 1989, wO Patent App. PCT/US1989/001,557. [online]. [cit. 2017-04-23].
URL https://www.google.com/patents/WO1989010254A1?cl=en

[6] Lengyel, E.: *Mathematics for 3D*. Massachusetts: Charles River Media, vyd. 1 vydání, 2004, ISBN 1584502770.

[7] Španěl, M.: *Základy počítačové grafiky: Osvětlení a stínování 3D objektů*, 2013.

[8] People.sc.fsu.edu: *MTL Files - Material Definitions for OBJ Files*. Retrieved 2010-11-26. [online]. [cit. 2017-04-23].
URL http://people.sc.fsu.edu/~jburkardt/data/obj/obj.html

[9] Phong, B. T.: *Illumination for computer-generated images*. Diplomová práce, The University of Utah, jul 1973.
URL http://www.dtic.mil/dtic/tr/fulltext/u2/a008786.pdf

[10] Phong, B. T.: *Illumination for Computer Generated Pictures. Commun. ACM*, ročník 18, č. 6, jun 1975: s. 311–317, ISSN 0001-0782, doi:10.1145/360825.360839, [online]. [cit. 2017-03-04].
URL http://www.cs.northwestern.edu/~ago820/cs395/Papers/Phong_1975.pdf

[11] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0, 609 s.

[12] Reichl, J.; Všetička, M.: *Eukleidovy postuláty*. online.
URL
http://fyzika.jreichl.com/main.article/view/1421-eukleidovy-postulaty

[13] Reichl, J.; Všetička, M.: *Použité metody převzorkování*. online.
URL http://fyzika.jreichl.com/main.article/print/1531-pouzite-metody-prevzorkovani

[14] Roth, S. D.: *Ray casting for modeling solids*. *Computer Graphics and Image Processing*, ročník 18, č. 2, 1982: s. 109 – 144, ISSN 0146-664X, doi:http://doi.org/10.1016/0146-664X(82)90169-1, [online]. [cit. 2017-03-17].
URL http://www.sciencedirect.com/science/article/pii/0146664X82901691

[15] Shreiner, D.; aj.: *OpenGL: průvodce programátora*. Computer Press, 2006, ISBN 80-251-1275-6, 679 s.

[16] Vojnar, T.: *Správa paměti*, 2015.

[17] Weisstein, E. W.: *Tessellation*. [online]. [cit. 2017-04-04].
URL http://mathworld.wolfram.com/Tessellation.html

Přílohy

Příloha A

Ovládání

Myš	
Vybrat kostku	Levé tlačítko
Rotace kamery	Pravé tlačítko
Přiblížit/ Oddálit	Kolečko

Tabulka A.1: Ovládání myši.

Numerická klávesnice			
Posun kostky vzad	2	Potvrdit místo kostky	Enter
Posun kostky vlevo	4	Přiblížit plochu	Plus
Rotace kostky	5	Oddálit plochu	Minus
Posun kostky vpravo	6	Posun kostky o úroveň výš	PageUp
Posun kostky vpřed	8	Posun kostky o úroveň níž	PageDown

Tabulka A.2: Ovládání numerické klávesnice.

Klávesnice			
Pohyb vpřed	W	Rotace nahoru	Šipka vpřed
Pohyb vzad	S	Rotace dolu	Šipka dolu
Pohyb vlevo	A	Rotace vlevo	Šipka vlevo
Pohyb vpravo	D	Rotace vpravo	Šipka vpravo
Nová kostka	N	Kostka $1 \times 1 \times 3$	1
Smazat vybranou kostku	R	Kostka $2 \times 1 \times 3$	2
Smazat všechny kostky	C	Kostka $3 \times 1 \times 3$	3
Zrušit označení kostky	ESC	Kostka $4 \times 1 \times 3$	4
Přepnout vybranou kostku	TAB	Kostka $2 \times 2 \times 3$	5
Nový soubor	Ctrl+N	Kostka $3 \times 2 \times 3$	6
Uložit	Ctrl+S	Kostka $4 \times 2 \times 3$	7
Načíst	Ctrl+L	Kostka $6 \times 4 \times 3$	8
Exportovat	Ctrl+E	Kostka $10 \times 4 \times 3$	9
Zpět	Ctrl+Z	Kostka $18 \times 4 \times 3$	0
Vpřed	Ctrl+Y		
Uložit jako...	Ctrl+Shift+S		

Tabulka A.3: Ovládání klávesnice.

Příloha B

Ukázka exportovaného souboru

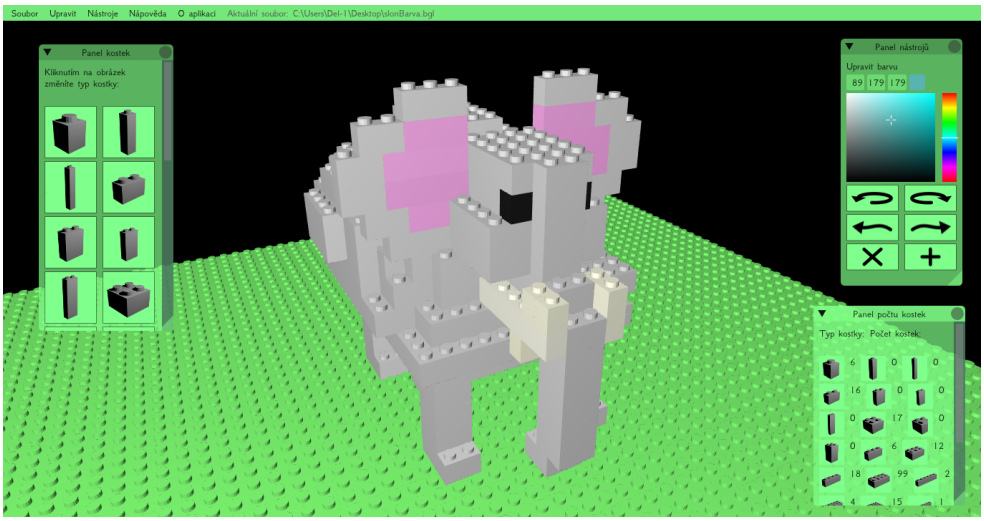
Exportovaný soubor obsahuje všechny potřebné kostky pro realizaci navrženého model. Každý řádek udává počet kostek jedinečné kombinace barvy a typu kostky.

Model slona

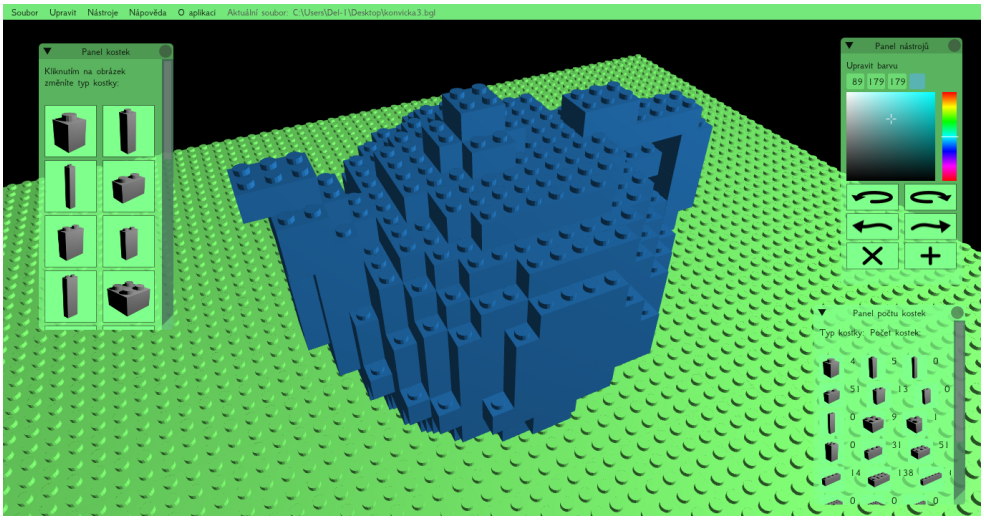
Typ kostky:	Barva kostky:	Počet:
10x1x3	#CBCBCBFF	2
1x1x3	#F7F7D9FF	1
1x1x3	#F7F7DDFF	5
2x1x3	#CBCBCBFF	16
2x2x3	#CBCBCBFF	17
3x1x3	#CBCBCBFF	4
3x1x3	#EE9BE0FF	1
3x1x3	#EE9DE1FF	1
3x2x3	#CBCBCBFF	12
4x1x3	#000000FF	1
4x1x3	#CBCBCBFF	11
4x1x3	#EC9CDFFF	1
4x1x3	#EE97E0FF	1
4x1x3	#F099E1FF	2
4x1x3	#F0A0E3FF	1
4x1x3	#F39EE4FF	1
4x2x3	#CBCBCBFF	99
6x1x3	#F7F6E5FF	1
6x1x3	#F7F7DFFF	1
6x2x3	#CBCBCBFF	4
6x4x3	#CBCBCBFF	15
8x1x3	#CBCBCBFF	1

Příloha C

Snímky aplikace



Obrázek C.1: Slon vymodelovaný v aplikaci BrickGL.



Obrázek C.2: Čajová konvička vymodelovaná v aplikaci BrickGL.